Regras de acessibilidade Métodos especiais referência this

Programação II

2015/16

Uso Packages

- Uma package é um conjunto de classes que partilham funcionalidades. ("pacote")
- O uso das classes duma package faz-se identificando a package e o nome.
  - java.util.Vector.size();

- O uso de cláusulas de importação dispensa a forma absoluta no nome da classe.
  - import java.util.Vector;
  - x.size();

Acessibilidade

- O java providencia mecanismos de controlo de acesso, às packages, classes, variáveis e métodos.
- A acessibilidade é estabelecida pelos modificadores de acesso: public, private, protected e o modificador por omissão.

Acessibilidade das Classes

- A acessibilidade duma classe é especificada usando na sua declaração algum modificador de acesso
  - Exemplo 1:
    - public class Ponto2D{...}
  - Exemplo 2:
    - class Ponto2D{...}

MODIFICADOR	ACESSO
public	todas
protected	-
private	-
nenhum	do package

Acessibilidade de métodos

- A acessibilidade dum método é especificada no header do método antes do tipo do retorno
- Exemplo 1:
  - ➡ public double getX(){...}
- Exemplo 2:
  - ➡ private aux(int x){...}

A API duma classe é constituída por todos os métodos da classe que não são privados

MODIFICADOR	ACESSO
public	qq classe
protected	pp classe, qq classe do package e qq subclasse
private	pp classe
nenhum	pp classe, classes do package

Acessibilidade de variáveis de instância

- Também as variáveis de instância sofrem das mesmas restrições de acessibilidade que os métodos:
  - public int x;
  - String s;
  - private y;
- Para garantir o encapsulamento as variáveis de instância não devem ser públicas

Overloading de métodos

- O java permite a existência dentro da mesma classe de métodos com o mesmo identificador. Este mecanismo é designado por "overloading" (sobrecarga) de métodos
- Como é descodificada a mensagem recebida por um objecto?
  - ➡ A assinatura do método, não contempla só o identificador
- Vantagem?
  - ➡ providenciar o mesmo identificador para métodos que realizem a mesma funcionalidade, mesmo que por outros meios.

Métodos Especiais

- É usual definir nas classes uma representação em String dos objectos criados.
- Qualquer método que devolva uma String é passível de ser usado, mas usemos um "standard"
- Método público que retorna uma String e que se chama toString, sem parâmetros

```
public String toString(){
/*código que define
a representação em String do objecto */
return  essaString; }
```

- Já falámos do toString() quando demos as Strings...

toString()

SE

é esperada uma String

e se existir na classe um método de assinatura

public String toString()

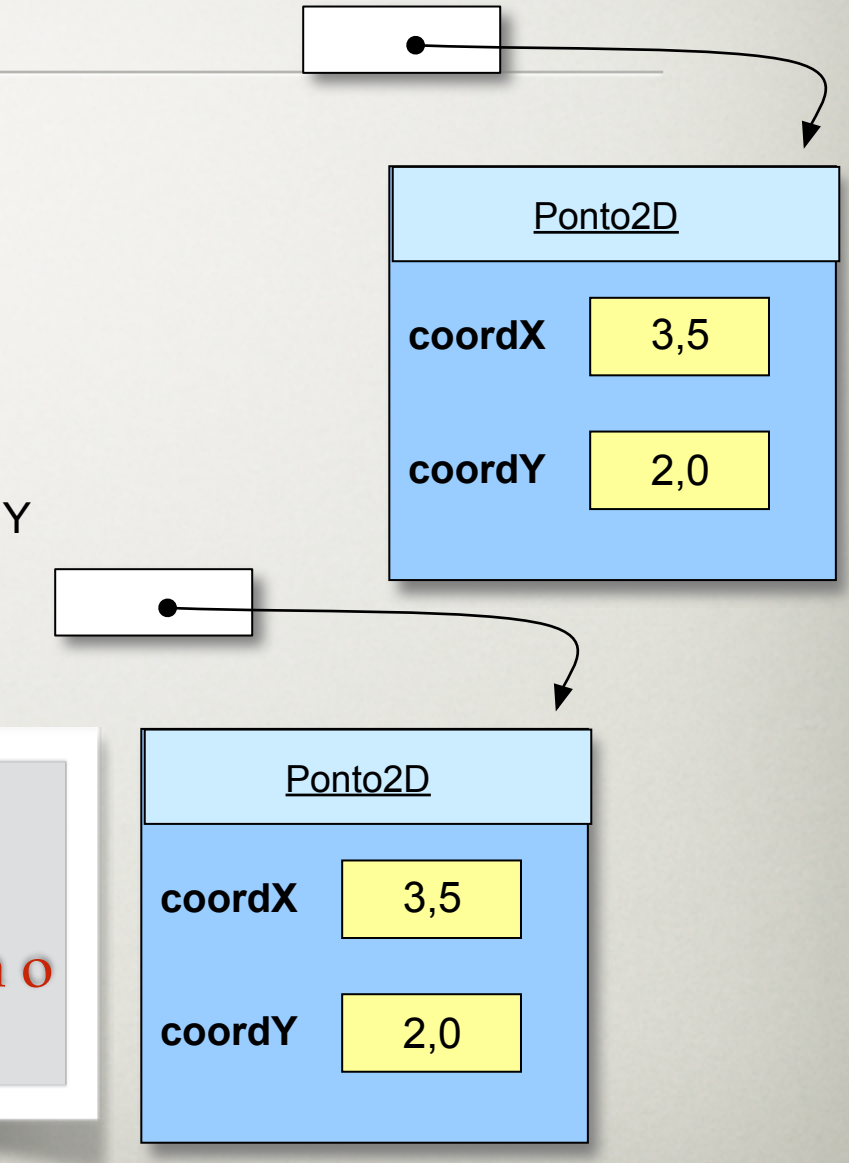
ENTÃO

toString é invocado automaticamente para converter em String o objecto

Igualdade de Objectos

```
Ponto2D X=new Ponto2D(3.5,2);
Ponto2D Y=new Ponto2D(3.5,2);
if (X==Y)
      System.out.println(“iguais");
else
     System.out.println("diferentes");
```

```
>diferentes
```

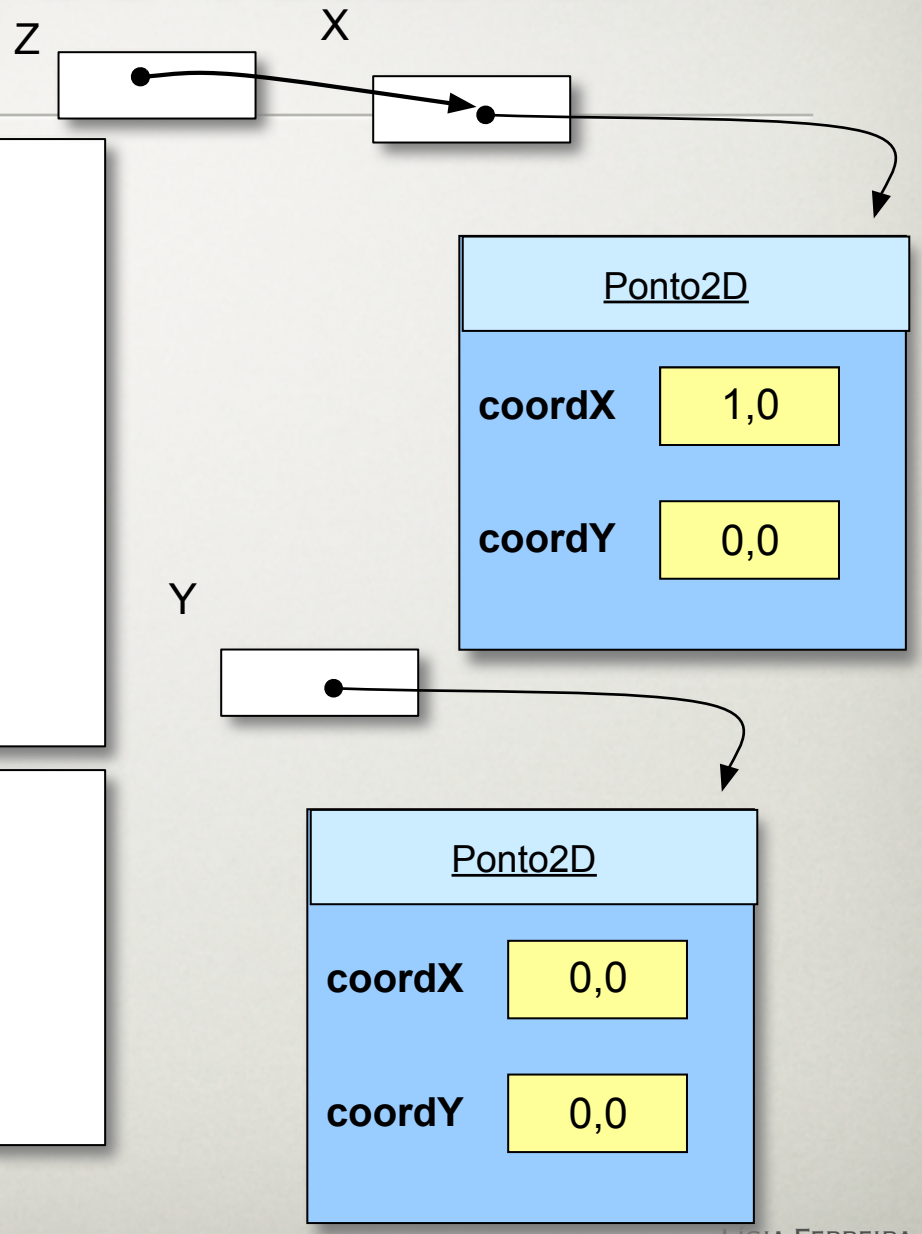


O operador == entre tipos primitivos significa o mesmo valor, mas para objectos significa o mesmo objecto

Igualdade de Objectos

```
Ponto2D X=new Ponto2D();
Ponto2D Y=new Ponto2D();
Ponto2D Z=X;
System.out.println(Z==X);
Z.setX(1);
System.out.println(X);
System.out.println(Y);
System.out.println(Z);
```

```
> true
> (1.0,0.0)
> (0.0,0.0)
> (1.0,0.0)
>
```



Métodos Especiais

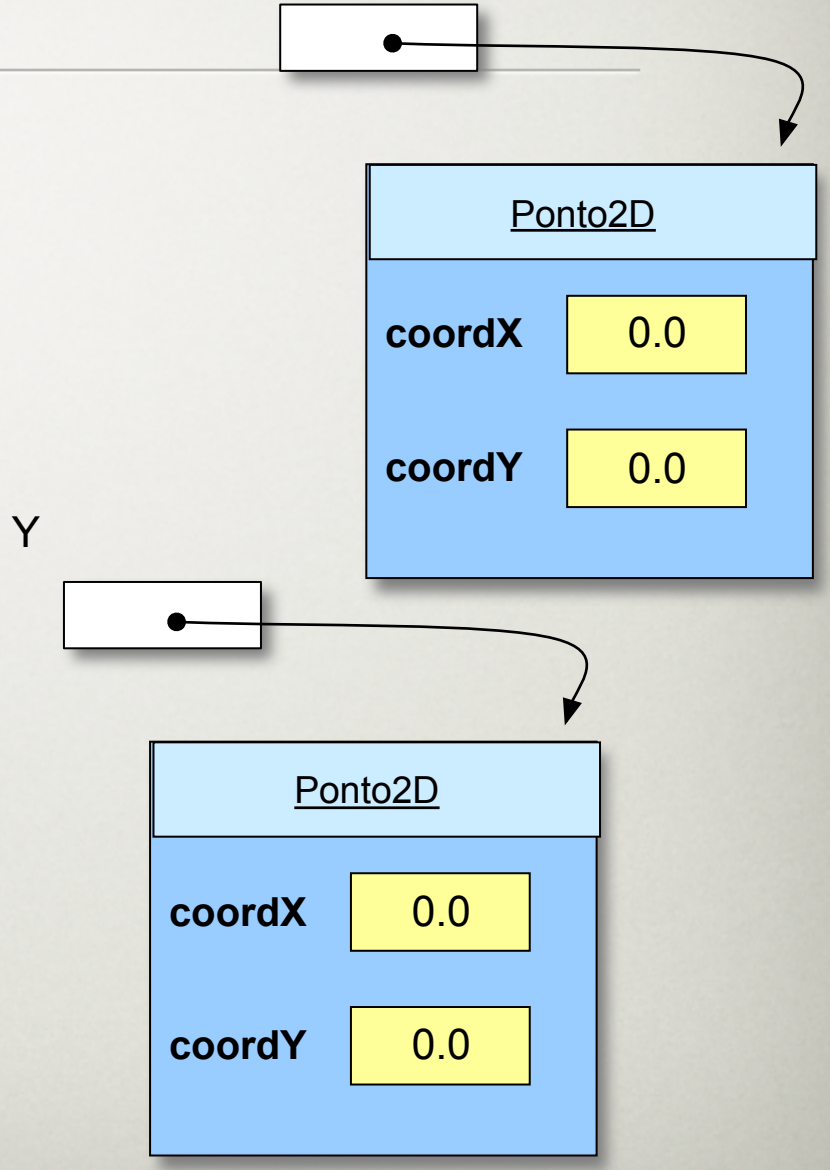
- Já sabemos que o == entre objectos permite saber se dois objectos são o mesmo, mas e se quisermos saber se dois objectos têm o mesmo valor?
- Dados 2 pontos2D podemos querer saber se são iguais:
  - ➡ Podemos definir a igualdade entre Pontos2D dizendo que são iguais se:
    - tiverem coordenadas iguais simultâneamente em X e Y

```
class Ponto2D{
.....
public boolean equals(Ponto2D x){
        return coordX==x.coordX &&
                coordY==x.coordY;
}
}
```

Igualdade entre objectos

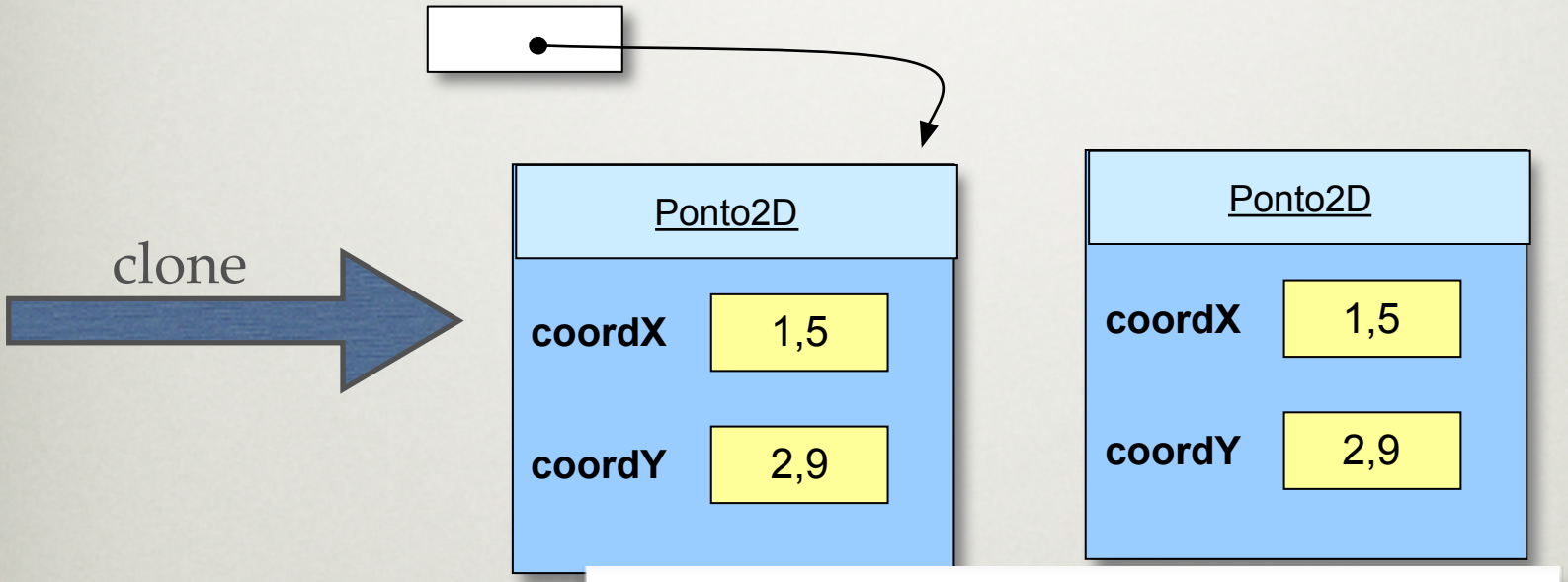
```
Ponto2D X=new Ponto2D();
Ponto2D Y=new Ponto2D();
System.out.println(X==Y);
System.out.println(X.equals(Y));
System.out.println(Y.equals(X));
```

```
> false
> true
> true
>
```



Clonagem

- Reproduzir um objecto exactamente igual a outro mas diferenciado, i.e., dado um objecto ao cloná-lo queremos outro objecto distinto do original mas exactamente igual



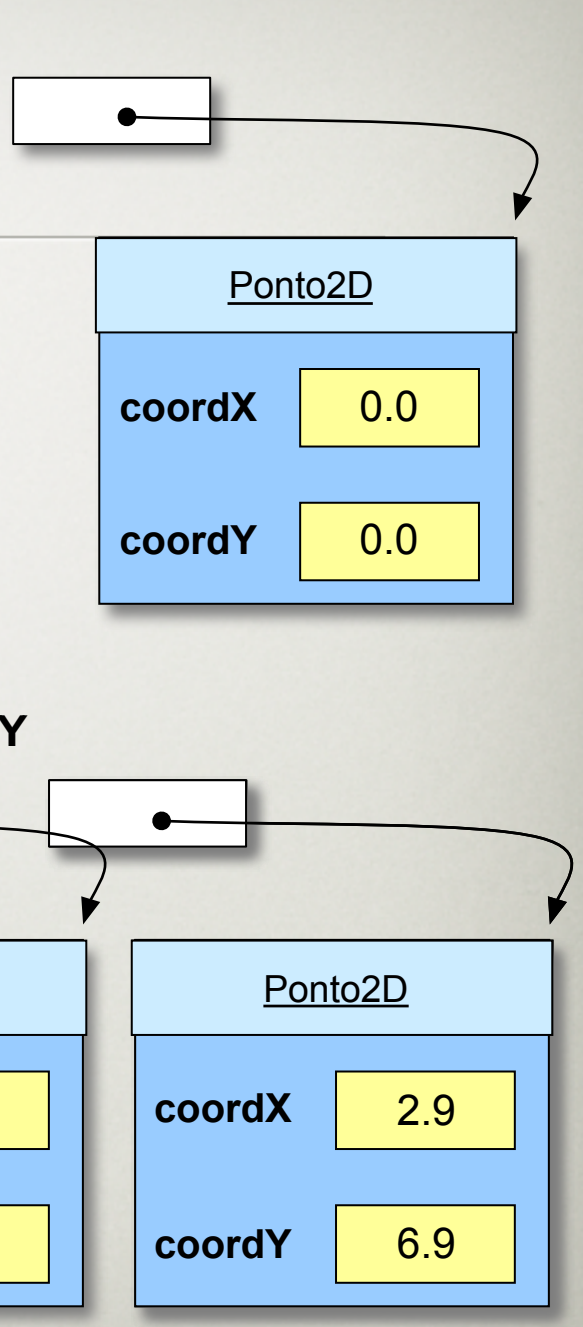
Se Y é um clone de X=>
X==Y é false e X.equals(Y) é true

Clonagem

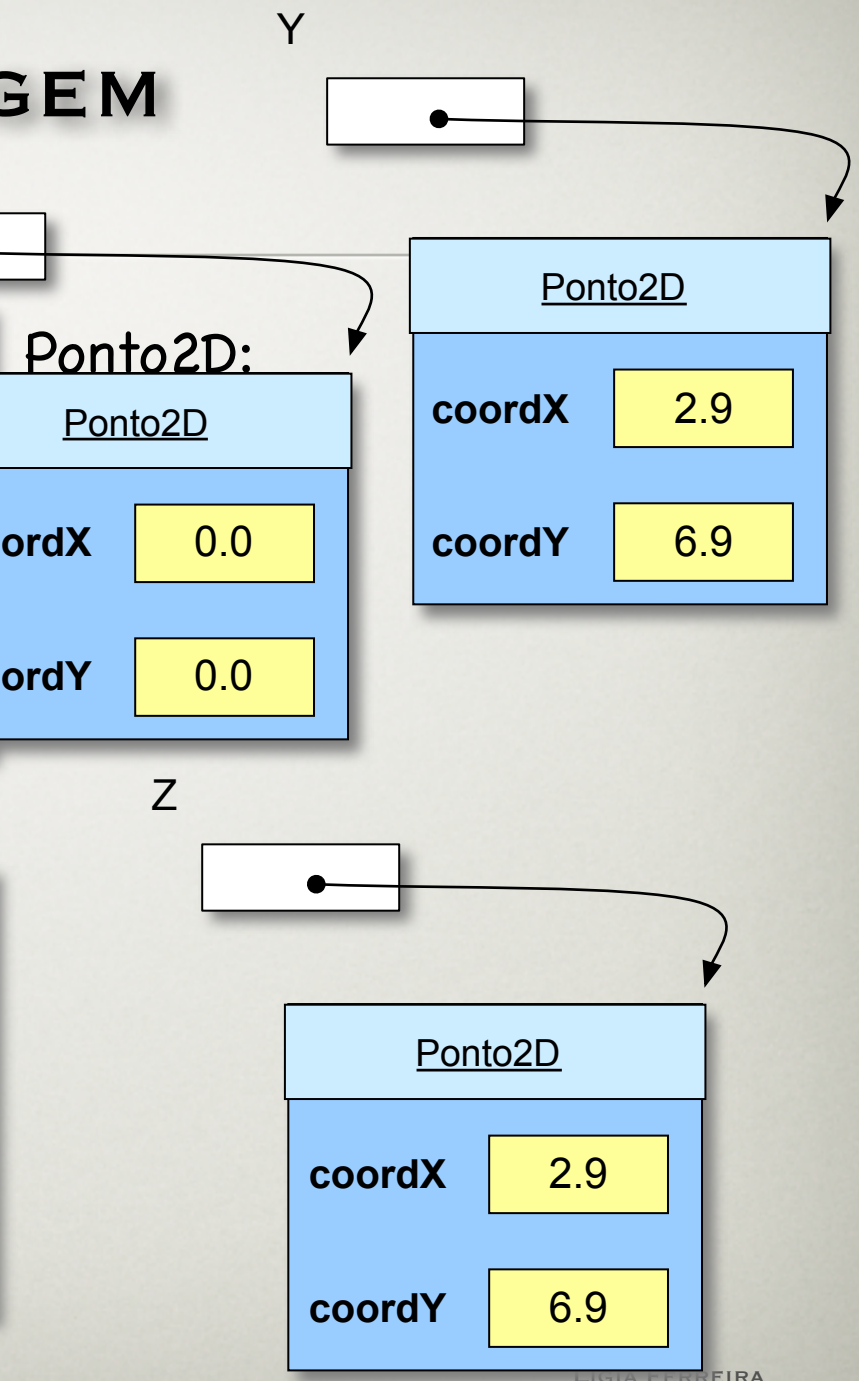
- Código para o método clone em Ponto2D:

```
class Ponto2D{
…..
public Ponto2D clone(){
      return new Ponto2D(coordX,coordY);
}
}
```

```
Ponto2D X=new Ponto2D();
Ponto2D Y=new Ponto2D();
Y.setX(2.9);
Y.setY(6.9)
Ponto2D z=Y.clone()
```



Clonagem




```
System.out.println("X="+X);
System.out.println("Y="+Y);
System.out.println("Z="+Z);
System.out.println(Y==Z);
System.out.println(Y.equals(Z));
```

```
> X=(0.0,0.0)
> Y=(2.9,6.9)
> Z=(2.9,6,9)
> false
> true
>
```

Referência this

- Dentro do código dum objecto, podemos referenciar o próprio objecto, ou enviar mensagens ao próprio objecto?
  - ➡ Usando a referência this
- Dentro do código de Ponto2D
  - coordX ou this.coordX
  - toString() ou this.toString()
  - Quando inequívoca a referência pode ser omitida (caso geral)

```
public void setX(double x){
    coordX = x;
}
```

variável de instância

```
public void setX(double coordX){
    this.coordX = coordX;
}
```

variável instância argumento

Lígia Ferreira
Salvador Abreu